Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 52 (2015) 996 – 1003

Procedia Computer Science

2nd International Workshop on Computational Antifragility and Antifragile Engineering (ANTIFRAGILE 2015)

A framework for trustworthiness assessment based on fidelity in cyber and physical domains

Vincenzo De Florio[a,*], Giuseppe Primiero[b]

[a]*MOSAIC, University of Antwerp & iMinds Research Institute, 2020 Antwerpen, Belgium*
[b]*Department of Computer Science, Middlesex University, London, UK*

Abstract

We introduce a method for the assessment of trust for *n*-open systems based on a measurement of fidelity and present a prototypical implementation of a complaint architecture. We construct a MAPE loop which monitors the compliance between corresponding figures of interest in cyber- and physical domains; derive measures of the system's trustworthiness; and use them to plan and execute actions aiming at guaranteeing system safety and resilience. We conclude with a view on our future work.

Peer-review under responsibility of the Conference Program Chairs
Keywords: Fidelity; trustworthiness; cyber-physical systems; *n*-open systems; resilience.

1. Introduction

Fidelity of an open system can be interpreted as the compliance between corresponding figures of interest in two separate but communicating domains, see[1]. In cyber-physical systems, perfect fidelity means that actions in the physical domain have a well-defined and stable counterpart in the cyber domain, and vice-versa. This is an ideal state, as no concrete cyber-physical system may guarantee at all times a perfect correspondence between its domains of action. In practice, fidelity is affected by circumstances that let the system drift from the optimal case. Our stance is that, by observing the characteristics of said drifting, we may introduce a fine-grained characterisation of the quality of system trustworthiness. To this aim, we introduce a practical method for the assessment of trust based on the measurement of fidelity in computational systems, including cyber-physical ones. As a way to measure fidelity drifting we propose to adopt and extend the approach described in[2,3]. We propose to make use of the accrued information to assess the characteristics of drifting in fidelity; we derive from it dynamic properties of both user and machine; evaluate it in terms of system's trustworthiness and use it to execute safety-assurance actions. This generates a trust-induced MAPE-loop.

* Corresponding author. Tel.: +32-3-2653905 ; fax: +32-3-2653777.
E-mail address: vincenzo.deflorio@gmail.com ().

Peer-review under responsibility of the Conference Program Chairs
doi:10.1016/j.procs.2015.05.092

The paper is structured as follows. In Sect. 2 we describe the conceptual model of system fidelity; in Sect. 3 the model for fidelity evaluation is implemented in an architecture for a cyber-physical system; in Sect. 4 we build the link to trustworthiness evaluation; finally, in Sect. 5 we conclude by drawing some observation and describing further lines of research.

2. Conceptual understanding of System Fidelity

In the present section we introduce a conceptual model of system fidelity, derived from the one presented in [1]. Our starting point is the concept of n-open systems (nOPS), characterised by the following properties:

- nOPS interact with one or more of the environments they are deployed in.
- nOPS base their action on the ability to sense n classes of raw facts taking place in their deployment environments.
- nOPS are able to construct and maintain n classes of internal representations of the raw facts, called *qualia*.
- *Qualia* are, to some extent, faithful, meaning that they timely reflect the dynamic variation of the corresponding class of raw facts.

We discuss here *fidelity* as a characterisation of the above-mentioned faithfulness. More formally, given any $S \in n\text{OPS}$, we consider n classes of raw facts, $[r]_i, 1 \leq i \leq n$, and n classes of binary operations, $[\dot{+}]_i$, such that each of the couples

$$([r]_i, [\dot{+}]_i) \tag{1}$$

constitutes an algebraic structure; for instance, when $[\dot{+}]_i$ is a singleton, then $([r]_i, [\dot{+}]_i)$ is a group. Likewise, for any $1 \leq i \leq n$, we call $[q]_i$ the class of *qualia* corresponding to $[r]_i$ and $[\oplus]_i$ the class of binary operations corresponding to $[\dot{+}]_i$. Moreover, as we did for (1), we assume that $([q]_i, [\oplus]_i)$ is an algebraic structure. Then, for each $1 \leq i \leq n$, we consider the following function:

$$\Phi_i : [r]_i \to [q]_i, \tag{2}$$

mapping the qualia corresponding to any raw fact in $[r]_i$. We refer to the Φ_i functions as the *reflective maps* of some n-open system S. Reflective maps are assumed to be bijective functions, with Φ_i^{-1} being the *inverted reflective maps* of S associating the raw fact corresponding to each input quale. We shall say that Φ_i expresses *perfect fidelity* between $([r]_i, [\dot{+}]_i)$ and $([q]_i, [\oplus]_i)$ if and only if Φ_i preserves its algebraic structures (i.e., it is an isomorphism). More formally, for any couple of raw facts $(r_1, r_2) \in [r]_i \times [r]_i$ and for all $+ \in [\dot{+}]_i$ and all $\cdot \in [\oplus]_i$: $\Phi_i(r_1 + r_2) = \Phi_i(r_1) \cdot \Phi_i(r_2)$.

Perfect fidelity may be better understood through an example. Let us assume that S is a cyber-physical system responsible for the operation of a mission critical hard-real-time service. An operator is responsible for the issuing of requests for service, which is done through a user interface (UI). A set of raw facts and prescribed behaviours pertaining to the physical environment are represented as "cyber-qualia" and "cyber-behaviours" stored in computer memories. Likewise, a set of "UI-qualia" and "UI-behaviours" are respectively rendered and operable through the UI. Perfect fidelity states that the correspondence between the physical, the cyber, and the UI domains is such that the prescribed behaviours as well as the referred raw facts and qualia are consistent on either of the involved domains. Thus certain operations and objects represented and rendered via the UI perfectly correspond to operations and objects encoded in S's computer components, which in turn perfectly correspond to physical actions having effects on physical entities. Obviously, perfect fidelity only represents a reference point and can not be sustained and guaranteed at all times in real life. A slightly different and more practical definition of fidelity is given by a function $\phi_i, 1 \leq i \leq n$:

$$\phi_i : [r]_i \to [q]_i, \quad \text{such that} \quad \forall + \in [\dot{+}]_i, \forall \cdot \in [\oplus]_i : \phi_i(r_1 + r_2) = \phi_i(r_1) \cdot \phi_i(r_2) \cdot \Delta_i(t). \tag{3}$$

As for the Φ_i function, ϕ_i returns the qualia associated with the input raw facts. Contrarily to the Φ_i function, the ϕ_i does not preserve their algebraic structures unless the value of the error component $\Delta_i(t)$ is zero. The use of lowercase "ϕ" is meant to suggest that ϕ_i represents a less-than-perfect version of Φ_i. The $\Delta_i(t)$ quantifies a drifting in time (represented by variable t) of the ability to create a trustworthy "internal" representation of an experienced raw fact.

In [1], fidelity is classified in function of the type of drifting. Classes may include, e.g., the following cases:

- Hard-bound fidelity drifting, exemplified by hard-real-time *n*OPS.
- Statistically-bound fidelity drifting—as typical of, e.g., soft real-time systems.
- Unbound fidelity drifting characterised by a "trend".
- Unbound fidelity drifting with a random trend.

Accordingly, very disparate cases can be presented to exemplify imperfect fidelity, e.g. the accidents experienced by the linear accelerator Therac-25[4,5] and the system failure caused by the last Scud fired during the Gulf War[6]. In the former case, one of the reasons that led to some of the accidents was that the UI-qualia and the cyber-qualia did not match when the operator was typing at a very fast pace. As a result of such imperfect fidelity, the quantities represented on the screen of the Therac-25 did not correspond to the data stored in its memories—and, regrettably, to the amount of radiations supplied to the patients by the linear accelerator. In the Patriot case, resulting in 28 US Army reservists being killed and 97 injured by a Scud missile on 25 February 1991. The missile-defence system was an *n*OPS that interacted with its environment through a number of context figures that included velocity and time. As discussed in[7], the cyber-quale corresponding to physical time was represented as the number of tenths of seconds from a reference epoch and stored in a 24-bit integer variable. Imprecision in the conversion of said variable into a real number translated in an *unbound drifting of fidelity over time*. The more the Patriot missile-defence system operated without a reboot, the larger was the Δ_i pertaining to time and, as a consequence, the greater the discrepancy between the expected and the real position and velocity of the incoming Scud missile. An obvious workaround to the above unbound drifting is that of rebooting the system regularly so as to rejuvenate[8] the qualia management system and bring back the Δ_i to "safe" values. Although both problem and workaround were known at the time of the accident, no upper bound was known beyond which the resilience of the system would be affected. Common belief was that the unresilience threshold would never be reached in practice. Regrettably, reality proved the trust on that belief to be misplaced. The Patriot missile that had to intercept the Scud never took off. The cases of the Therac-25 and of the Patriot system reveal a common denominator: behaviours such as those of a human operator or those produced by a numerical algorithm are all translated into a same, homogeneous form: that of a stream of numerical data representing samples of the $\Delta_i(t)$ dynamic systems. A major methodological assumption in the present work is that the above data could be compared with other data representing reference conditions. In the Therac-25 case, such data may correspond to, e.g., reference user stereotypes of expected operator behaviours, represented for instance as the numerical weights in a Hidden Markov Model[13]. Likewise, in the Patriot case, those reference data may correspond to, e.g., a threshold representing safe ranges for accumulated or cumulative numerical errors produced through the iterations of numerical methods. In both the exemplified cases, assessing fidelity is thus translated into the problem of evaluating a "distance" between observed and reference data.

In what follows, we propose an architecture and a prototypical system to evaluate systematically a system's fidelity drifting. These are modelled on a toy example, easily modified and extended to a real case scenario, whose presentation we reserve to an extended version of this work.

3. Janus' Architecture

In view of the above discussion, our approach to systematic evaluation of fidelity requires at least the following components:

- A sensory service, interfacing the deployment environments so as to register a number of "raw facts", namely variations in a number of environmental properties. Raw facts could refer, for instance, to variations in luminosity, temperature, or the amount of network bandwidth available between two endpoints.
- A uniform qualia service, providing consistent, timely and reliable access to cyber-qualia (computer-based representations of the raw facts).
- An application layer, providing a convenient means for fidelity assessment and reactive control.

In what follows we introduce the above components and a prototypical system compliant to the just sketched models, see Fig. 1.

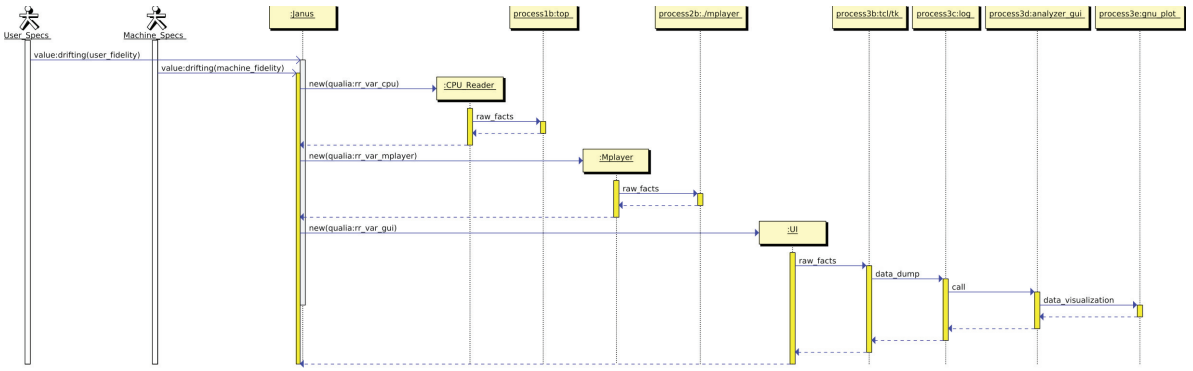

Fig. 1. Sequence diagram of the Janus' components.

3.1. Sensory and Qualia Services

Our sensory and qualia service is based on so-called reflective and refractive variables (RRvars) [10,11], a tool for the development of *n*OPS in the C programming language. The idea behind RRvars is quite simple: a number of predefined variables provide access to the qualia of corresponding raw facts. Those variables are "volatile", meaning that their content is asynchronously and continuously updated by an associated thread. Thus for instance RRvar `int cpu` does not retain a constant value; rather, it is continuously updated by the `Cpu()` thread. Such thread cyclically retrieves the percentage of CPU currently in use and stores it in the memory cells associated to `cpu` as an integer number ranging between 0 (CPU fully available) and 100 (no CPU available). In the current implementation, which runs on Linux and Windows/Cygwin systems, `Cpu()` retrieves its raw facts by calling the `top` utility. This is referred to as `process1b` in Fig. 1.

A second and slightly more complex example is given by RRvar `int mplayer`, a variable updated by thread `Mplayer()`, referred to as `process2b` in Fig. 1. The latter component communicates with an instrumented MPlayer movie player [12] through a simple UDP client/server protocol. By reading the content of `mplayer` one is informed of the state of the MPlayer—see Fig. 3. Currently, the following integer values are used:

```
#define UDPMSG_STOP     1 // mplayer has finished playing a video
#define UDPMSG_SLOW     2 // mplayer is encountering problems while playing a video
#define UDPMSG_PAUSED   3 // mplayer has been paused
#define UDPMSG_START    4 // mplayer has been launched
#define UDPMSG_SIGNAL   5 // mplayer caught an exception and is about to exit abnormally.
```

A third case is given by RRvar `int ui`, updated by thread `Ui()`. This is a special case in that this RRvar represents a UI-qualia (see Sect. 2) reporting raw facts specific of an instance of a user interface. This is referred to as `process3b`–`process3e` in Fig. 1. Said user interface and the `Ui()` thread communicate transparently of the operator via the same mechanism presented for RRvar `mplayer`. The values returned in RRvar `int ui` represent usability raw facts derived by comparing the behaviours exercised by the current user with "reference behaviours" representing the expected behavioural patterns of a trustworthy operator. The method to derive these raw facts is described in [2,13]. This method may be used to detect gradual behavioural driftings (due to, e.g., fatigue, stress, or the assumption of psychotropic substances) and sudden behavioural driftings (caused, e.g., by an account takeover or other cyber-criminal attacks).

```
/* File Janus.c. Created/modified on Fre Feb 27 10:44:31 CET 2015 */
#include "rrvars_init.h"   // RRvars initialization routines
int main (int argc, char *argv[])
{
   int cpuold, mplayerold, uiold;
   RR_VARS                 // initializes the RRvars client
   RR_VAR_CPU              // spawns and initializes thread Cpu()
   RR_VAR_MPLAYER          // spawns and initializes thread Mplayer()
   RR_VAR_UI               // spawns and initializes thread Ui()
   printf("cpu == %d\nmplayer == %d\nui == %d\n", cpu, mplayer, ui);
   while (1) { sleep(1);
       if (cpuold != cpu) printf("cpu == %d\n", cpuold=cpu);
       if (mplayerold != mplayer) printf("mplayer == %d\n", mplayerold=mplayer);
       if (uiold != ui) printf("ui == %d\n", uiold=ui);
   }
}
#include "rrvars_end.c"   // RRvars termination routines
```

Fig. 2. Typical structure of an RRvar client. Here three RRvars (cpu, mplayer, and ui) are declared and continuously displayed.

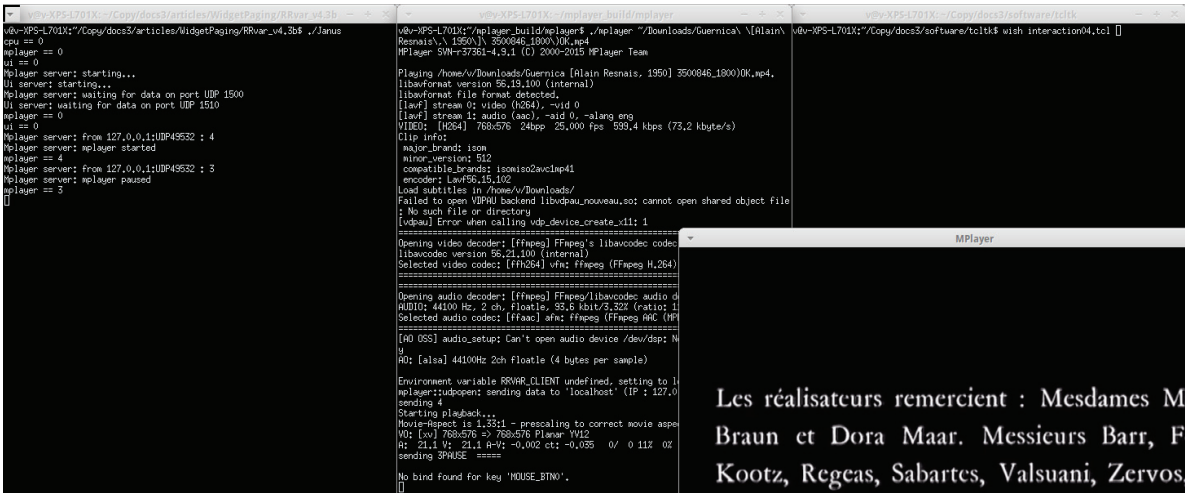

Fig. 3. An instance of the MPlayer connects with Janus and reports its state.

3.2. Control Layer: Janus

Janus is the name of our exemplary RRvar client component.[2] The structure of `Janus` is the one typical of RRvar clients [11] and exemplified in Fig. 2. As can be seen from the picture, the RRvar metaphor makes it possible to quickly define *n*OPS components based on the three classes of qualia presented above, see Fig. 4.

4. Fidelity as Trustworthiness

In this final section, we link the model of open systems offered in Sect. 2, and their architecture implemented in Sect. 3, to a model of trustworthiness evaluation. We use fidelity to assess the working conditions of an open system and to establish a metric of trustworthiness. Our final objective is to provide a qualitative extension of the standard MAPE-loop based on trustworthiness assessment. Trust for security, management and reputation systems is gaining a lot of attention in the literature and it is typically accounted for as a first-order relation between (possibly autonomous) agents or system's modules, see e.g. [16,17,18,19,20]. Our account considers trust as a second order property characterising cumulatively system's fidelity, where the latter is obtained as the dynamic variation of properties of the system's modules. This approach to second-order trust has been already used for information transmission evaluations ([21]), access-based control ([22]), and software management systems ([23]). In the present analysis, trustworthiness is used to plan reaction to malfunctioning and restoring of functionalities in cyber-physical systems. As the cases of Therac-

[2] As for the system described in [14], the name of our component comes from mythical *Janus Bifrons*, the god of transitions, who had two faces and thus could observe and reason by considering two different "views" at the same time. Of the proposed etymologies of Janus, particularly intriguing here is the one proposed by Paul the Deacon [15]: hiantem, hiare, "to be open". Due to this fact one would be tempted to refer to Janus Bifrons here as to a *2-open system*.

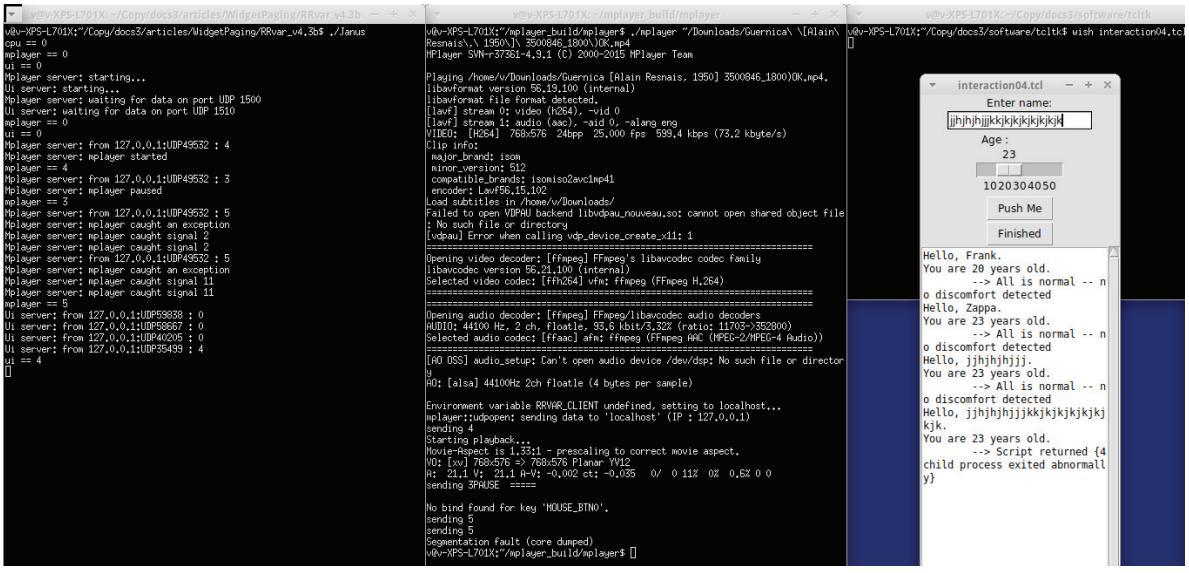

Fig. 4. The RRvar client connects with both MPlayer and an exemplary user interface.

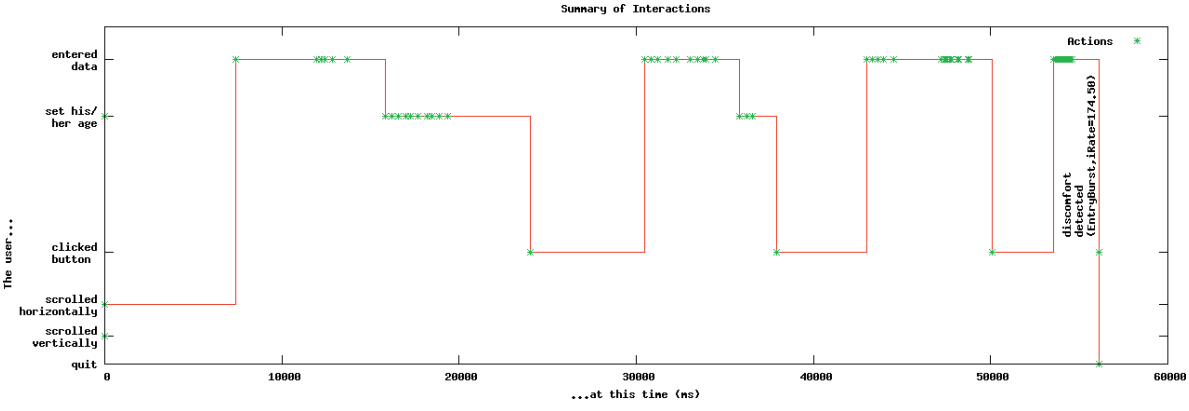

Fig. 5. Log of the interaction between user and UI. Between 50 and 60s a rapid burst of keystrokes is interpreted as an anomalous situation.

25 and the Patriot-missile defence system show, unacknowledged drifting can be crucial in maximising unjustified trust to dangerous levels. On the other hand, a metric evaluating minimal functionality thresholds can minimise unjustified mistrust, reducing confidence that the system *will* choke and eventually fail (antitrust). Early cases of low true-alarm rate or high false-alarm rate in automation do not need to set constants for future behaviour of the user. Similarly, criteria to compare intended and current behaviour are essential to allow mechanical assessment of user's inability, incompetence or threats. Fidelity and drifting can provide the systematic methodology and quantitative model to continually evaluate and experiment the system's vulnerability in the context of its operations, thus making the dichotomy conditional vs. unconditional trust (faith) working. Trust can be thought of as a measure of confidence about minimal drifting from fidelity for all the components, and hence that the overall behaviour of the system is sufficiently resilient. In the following, we reconstruct the process of fidelity monitoring, trustworthiness evaluation and operation execution in terms of a MAPE-loop designed for trust, see [24]. In the architecture presented in Sect. 3, the *Janus* assesses the behaviour of the system. This is parametrised in view of reflective variables for CPU consumption and a component's operations (on the machine side of the system), and for the user interface (on the user's side). Mapping of these variables values as raw facts and qualia provides a measure of system's fidelity:

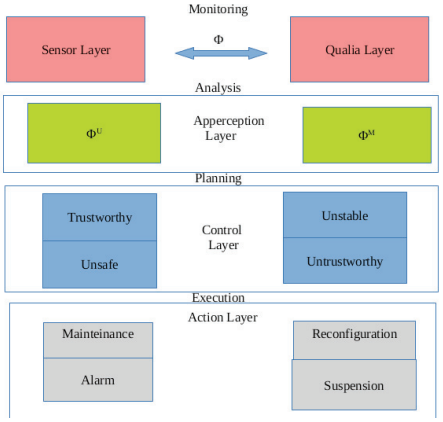

Fig. 6. A MAPE-loop for System trustworthiness based on fidelity.

- $\Phi_{CPU} : [r]_{CPU} \rightarrow [q]_{CPU}$, obtained by the mapping of input values from the *top* process to pre-selected parameters assigned to CPU-consumption behaviour;
- $\Phi_{component} : [r]_{component} \rightarrow [q]_{component}$, obtained by the mapping of input values from the executable's observable behaviour to pre-selected parameters assigned to its operations;
- $\Phi_{UI} : [r]_{UI} \rightarrow [q]_{UI}$, obtained by the mapping of input values from the user's observable behaviour to pre-selected parameters assigned to a standard or expected user's behaviour.

The system's component monitoring the classes $[r]_i$ of raw facts is called the *sensor layer*; similarly, we use *qualia layer* to refer to the component monitoring the classes $[q]_i$ of qualia. The combination of the sensory and representative layers constitutes the *Monitoring* component within our MAPE-loop. Fidelity is then approximated as the inversely proportional function of the drifting from appropriate mappings Φ_i. We shall refer to the value of user-based mappings as *user-defined fidelity* (Φ^U); correspondingly, we shall call *machine-defined fidelity* (Φ^M) the value based on mappings related to the machine behaviour. For the *Janus* introduced in Sect. 3,

$$\Phi^U = 1/\Delta(t)_{UI} \tag{4}$$

$$\Phi^M = 1/f(\Delta(t)_{CPU}, \Delta(t)_{exec}) \tag{5}$$

where f is some function, weighted according to domain-specific and user defined parameters. We refer to the set of values $\mathbf{\Phi}(t) = \{\Phi^U, \Phi^M\}$ as the content of our *Analysis* component, with the global value $\mathbf{\Phi}$ parametrised by time. As an example, consider the class of mappings Φ_{UI}, with a value of the sensor layer indicating e.g. quick typing and a value of the qualia layer returning a distress indication: in this case the fidelity layer reports an high value. As an example across distinct mappings, assume that the reflective variable for MPlayer indicates that the application is running slower, while the one for CPU monitoring indicates low usage value: this is expressed by a low fidelity value across the two classes in Φ^M. Analysing fidelity values across the distinct monitoring layers allows a cumulative evaluation to be obtained. This value is monitored by the *apperception layer*. This layer is used to evaluate system trustworthiness as a global value of user-defined and machine-defined fidelity values. The next level is represented by the *Janus* feeding the content of the *apperception layer* into the *control layer*. This corresponds to the *Planning* component of our system. At this stage, system trustworthiness is matched to a resilience scale that identifies and automatically triggers actions aimed at preserving system safety or enabling ameliorating conditions. The latter part of the system is the *Execution* component, monitoring an *action layer*. Despite the fact that a resilience scale should be highly domain specific, a possible general model can be given in terms of four essential stages:

1. *Trustworthy System* identifies high levels of $\mathbf{\Phi}(t)$, inducing optimal, sustainable working conditions;
2. *Unstable System* identifies high-to-medium Φ^U and low Φ^M levels, inducing reconfigurable working conditions;
3. *Unsafe System* identifies high-to-medium Φ^M and low Φ^U levels, inducing alarm-rising working conditions;

4. *Untrustworthy System* identifies low-levels of $\mathbf{\Phi}(t)$, inducing inadvisable or below-safety working conditions.

The above analysis is summarised in the MAPE-loop in Fig. 6.

5. Conclusions

We have presented a prototypical model architecture for a trust-based MAPE-loop for cyber-physical systems. The trust evaluation is grounded on the assessment of fidelity drifting with respect to values representing ideal reference conditions for both user and machine. Although prototypical, we believe that our architecture proves the feasibility of our approach to trustworthiness assessment and paves the way towards future implementations. Our goal is to develop systems that are trustworthy in integrating quality-of-service and quality-of-experience, by optimising the relations between system-level, "microscopic" aspects and user-level, "macroscopic" ones. A further goal is to extend our architecture into that of a MAPE-K loop and apply machine learning methods such that systems based on our approach may systematically improve the match with the environments they interact with. Costs analysis for the trust-based MAPE-loop remains to be explored. Its use at early stages of design can reduce risks; our examples shows, nonetheless, that it is possible to deploy the methodology on existing software, by selecting relevant variables. Further work will focus on formal modelling of trustworthiness assessment in a probabilistic epistemic setting.

References

1. De Florio, V. Antifragility = elasticity + resilience + machine learning. Models and algorithms for open system fidelity. *Procedia Computer Science* 2014;**32**:834–841. URL: `http://goo.gl/Ji40dH`. 1st ANTIFRAGILE workshop (ANTIFRAGILE-2014).
2. De Florio, V., Blondia, C. Safety enhancement through situation-aware user interfaces. In: *System Safety, incorporating the Cyber Security Conference 2012, 7th IET International Conference on*. 2012, p. 1–6. doi:10.1049/cp.2012.1520.
3. De Florio, V., Blondia, C. Reflective and refractive variables: A model for effective and maintainable adaptive-and-dependable software. In: *Proc. of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2007)*. Lübeck, Germany; 2007.
4. Leveson, N., Turner, C.S. An investigation of the Therac-25 accidents. *IEEE Computer* 1993;**26**(7):18–41.
5. De Florio, V. Software assumptions failure tolerance: Role, strategies, and visions. In: *Architecting Dependable Systems VII*; vol. 6420 of *LNCS*. Springer; 2010, p. 249–272. URL: `http://dx.doi.org/10.1007/978-3-642-17245-8_11`.
6. Schmitt, E. After the war; army is blaming patriot's computer for failure. *New York Times* 1991;URL: `http://goo.gl/uGNhkm`.
7. Grottke, M., Trivedi, K.S. Fighting bugs: Remove, retry, replicate, and rejuvenate. *IEEE Computer* 2007;**40**(2):107–109.
8. Grottke, M. et al. The fundamentals of software aging. In: *Proc. of the 1st Int.l Workshop on Software Aging & Rejuvenation*. 2008.
9. Anonymous. Patriot missile defense: Software problem led to system failure at Dhahran, Saudi Arabia. Tech. Rep. GAO/IMTEC-92-26; U.S. Government Accountability Office; 1992. URL: `http://www.gao.gov/products/IMTEC-92-26`.
10. De Florio, V., Blondia, C. On the requirements of new software development. *Int.l J. of Business Intelligence & Data Mining* 2008;**3**(3).
11. De Florio, V., Blondia, C. Reflective and refractive variables: A model for effective and maintainable adaptive-and-dependable software. In: *Proc. of the 33rd Euromicro Conf. on Soft. Eng. & Adv. App. (SEEA 2007)*. Lübeck, Germany: IEEE Comp. Soc.; 2007.
12. Anonymous. Mplayer — the movie player. 2015. Retrieved on February 3, 2015 from www.mplayerhq.hu/design7/info.html.
13. Duyse, J.V. *A toolkit for the concurrent analysis and adaptation of graphical user interfaces*. Master's thesis; Dept. Math. & Comp. Sci., Univ. of Antwerp, Belgium; 2013. URL: `http://tinyurl.com/k5a8zyo`; promotor: V. De Florio.
14. De Florio, V. et al. A hypermedia distributed application for monitoring and fault-injection in embedded fault-tolerant parallel programs. In: *Proc. of the 6th Euromicro Workshop on Parallel & Distrib. Processing (Euro-PDP'98)*. Madrid, Spain: IEEE Comp. Soc.; 1998, p. 349–355.
15. Paulus Diaconus. *Excerpta ex libris Pompeii Festi de significatione verborum*. W. M. Lindsay; 1930.
16. Carbone, M., Nielsen, M., Sassone, V. A formal model for trust in dynamic networks. In: *1st Int.l Conf. on SW Eng. and Formal Methods (SEFM 2003), 22–27 September 2003, Brisbane, Australia*. IEEE Comp. Soc.. 2003. doi:10.1109/SEFM.2003.1236207.
17. Clarke, S. et al. Trust*: Using local guarantees to extend the reach of trust. In: *Security Protocols XVII, 17th Int.l Workshop, Cambridge, UK, April 1–3, 2009. Revised Selected Papers*; vol. 7028 of *LNCS*. Springer. ISBN 978-3-642-36212-5; 2009, p. 171–178.
18. Chang, J. et al. AS-TRUST: A trust quantification scheme for autonomous systems in BGP. In: Proc. of *Trust and Trustworthy Computing - 4th Int.l Conf.l, TRUST 2011, Pittsburgh, PA, June 22-24, 2011*; vol. 6740 of *LNCS*. Springer. ISBN 978-3-642-21598-8; 2011, p. 262–276.
19. Grandison, T., Sloman, M. A survey of trust in internet applications. *Commun Surveys Tuts* 2000;**3**(4):2–16.
20. Yan, Z., Prehofer, C. Autonomic trust management for a component-based software system. *IEEE Transactions on Dependable and Secure Computing* 2011;**8**(6):810–823. doi:http://doi.ieeecomputersociety.org/10.1109/TDSC.2010.47.
21. Primiero, G., Taddeo, M. A modal type theory for formalizing trusted communications. *J Applied Logic* 2012;**10**(1):92–114. URL: `http://dx.doi.org/10.1016/j.jal.2011.12.002`. doi:10.1016/j.jal.2011.12.002.
22. Primiero, G., Raimondi, F. A typed natural deduction calculus to reason about secure trust. In: *2014 12th Annual Int.l Conf. on Privacy, Security and Trust, Toronto, ON, Canada, July 23-24, 2014*. IEEE. 2014, p. 379–382.
23. Boender, J., Primiero, G., Raimondi, F.. Minimizing transitive trust threats in software management systems. Tech. Rep.; Foundations of Computing Group, Middlesex University; 2015.
24. Jacob, B. et al. *A Practical Guide to the IBM Autonomic Computing Toolkit*. IBM Redbooks; 2004.